WLKATA MIROBOT Python SDK使用手册

版本历史

版本号	更新时间	变更记录	编写人
1.0	2021-10-09	文档初始化	邢顺凯(阿凯)
1.1	2021-11-17	添加了传送带的适配	邢顺凯(阿凯)
1.2	2022-04-02	修改部分案例跟代码	邢顺凯(阿凯)

录目

WLKATA MIROBOT Python SDK使用手册

版本历史

目录

Mirobot 坐标系定义

安装Mirobot Python SDK

创建机械臂对象

机械臂归零(Homing)

API-home 机械臂同时归零

API-home_1axis 单轴归零

API-home_slider 滑台归零

示例脚本-机械臂归零

限位(Limit)

API-set_hard_limit 设置硬限位

API-set_soft_limit 设置软限位

各轴解锁(Unlock All Axis)

API-unlock_all_axis 各轴解锁

设置默认运动速度(Set Speed)

API-set_speed 设置默认运动速度

机械臂回零(Go To Zero)

API-go_to_zero 机械臂回零

设置关节角度(Set Joint Angle)

API-set_joint_angle

示例脚本-设置机械臂关节角度

滑台控制(Slider Control)

API-set_slider_posi 设置滑台位置

示例脚本-滑台位置控制

传送带控制(Conveyor Control)

API - set_conveyor_range 设置传送带范围

API-set_conveyor-posi 设置传送带位置

示例脚本-传送带控制

工具-选择末端工具

API-WlkataMirobotTool末端工具

API-set_tool_type 选择末端工具

示例脚本-设置末端工具类型

工具-设置工具偏移量

API-set_tool_offset 设置工具偏移量 示例脚本-设置工具的偏移量 工具-气泵 API-pump_suction 气泵吸气 API- pump_blowing 气泵吹气 API-pump_off 气泵关闭 示例脚本-气泵控制 工具-舵机夹爪 API-gripper_open 夹爪打开 API-gripper_close 夹爪闭合 API-set_gripper_spacing 设置夹爪间距 示例脚本-机械臂爪相关API测试 机械臂状态信息 API-get_status - 获取并更新机械臂状态 机械臂状态符 关节角度 工具位姿 气泵状态 运动模式 示例脚本-获取机械臂状态 工具-末端姿态 API-set_tool_pose 示例脚本-机械臂工具位姿控制 轨迹规划-点到点(P2P) API-p2p_interpolation 示例脚本-点到点快速移动 轨迹规划-直线插补 API-linear_interpolation 线性插补 示例脚本-直线插补 轨迹规划-圆弧插补 API-circular_interpolation 圆弧插补 示例脚本-圆弧插补 轨迹规划-门式轨迹规划 API-set_door_lift_distance 设置抬起高度 API-door_interpolation 门式轨迹插补

Mirobot 坐标系定义

示例脚本-门式插补



Fig. 1 The six axes of Mirobot



Fig. 2 Robot coordinate system and origin

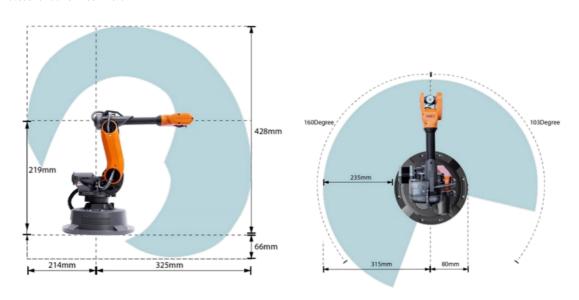
Mirobot六自由度机械臂关节(轴)定义如左图所示, 关节角度单位为°。 另外滑台/传送带属于第七轴,位置单位为mm。

机械臂的机械臂基坐标系如右图所示。



工具坐标系定义:默认机械臂第6关节末端法兰盘底面中心点为工具坐标原点,如果安装其他工具,则可以根据需要选取工具上某一点为参考点,并设定其相对于法兰盘中心的 XYZ 偏移量,则笛卡尔坐标模式下控制的机械臂末端位姿即为该点位姿。

机械臂末端的工作空间:



安装Mirobot Python SDK

Mirobot Python SDK的代码仓库的名字叫做 wlkata-mirobot-python

Github代码仓库

Windows操作系统, 通过pip3安装mirobot python sdk

```
1 pip install wlkata-mirobot-python
```

Ubuntu</mark>操作系统, 通过pip3安装mirobot python sdk

```
1 | sudo pip3 install wlkata-mirobot-python
```

创建机械臂对象

导入依赖

```
from wlkata_mirobot import WlkataMirobot
```

创建机械臂对象

```
1 | arm = WlkataMirobot()
```

默认为自动寻找设备模式,扫描串口设备并尝试通信,如果得到机械臂状态信息则创建链接。

如果想要手动设定mirobot的设备号,可以传入端口号 portname

Windows平台下端口名称格式为 COM + 编号, 示例如下:

```
1  # - Windows: COM + ID, exp: 'COM3'
2  arm = WlkataMirobot(portname='COM3')
```

Linux平台下端口名称格式为 /dev/ttyUSB + 编号, 示例如下:

```
1 | arm = WlkataMirobot(portname='/dev/ttyUSB0')
```

本文后续 arm 均指代mirobot对象.

机械臂归零(Homing)

机械臂上电的时候, 或者刚开始建立串口连接的时候, 机械臂的各个轴处于锁定状态。 必须执行机械 臂归零的动作, 才能解锁各轴, 然后才能执行动作。

API-home 机械臂同时归零

机械臂本体或者 机械臂本体+滑台同时归零。

函数原型

```
1 def home(self, has_slider=False):
```

输入参数

• has_slider: bool 是否有滑轨

使用示例

默认值为 False, 默认只有机械臂本体, 直接执行

```
1 | arm.home()
```

如果是有滑台的情况, 需要将 has_slider 参数设置为 True

```
1 arm.home(has_slider=True)
```

API-home_1axis 单轴归零

单轴归零

函数原型

```
1 def home_laxis(self, axis_id):
```

输入参数

• axis_id: int 轴的编号 取值范围 [1,7]

使用示例

```
1  # 关节1归零
2  arm.home_1axis(1)
```

API-home_slider 滑台归零

滑台单独归零

函数原型

```
1 | def home_slider(self)
```

使用示例

```
1 arm.home_slider()
```

示例脚本-机械臂归零

```
1 '''
2 机械臂回归机械零点与状态查询
3 '''
4 from wlkata_mirobot import WlkataMirobot
```

```
5
   import time
   print("实例化Mirobot机械臂实例")
   arm = WlkataMirobot()
10 # 机械臂Home 多轴并行
11
   print("机械臂Homing开始")
12
13 # - 一般情况, 在无第七轴的时候, 直接执行 arm.home() 即可,
14  # 参数has_slider默认为False。
15 # - 如有有滑台(第七轴), 将has_slider设置为True
16 arm.home()
   # arm.home(has_slider=False)
17
18 # arm.home(has_slider=True)
   print("机械臂Homing结束")
20
21 # 状态更新与查询
22 print("更新机械臂状态")
23 arm.get_status()
24 print(f"更新后的状态对象: {arm.status}")
25 print(f"更新后的状态名称: {arm.status.state}")
```

限位(Limit)

机械臂运动触发软限位后, 机械臂停止运动, 此时反方向运动即可解除; 机械臂触发硬限位后机械臂锁死, 需重启机械臂;

API-set_hard_limit 设置硬限位

设置是否开启限位开关的硬限位

函数原型

```
1 | def set_hard_limit(self, enable):
```

输入参数

• enable: bool 使能

True: 开启硬限位False: 关闭硬限位

```
1 arm.set_hard_limit(True)
```

API-set_soft_limit 设置软限位

设置是否开启软限位

函数原型

```
1 def set_soft_limit(self, enable)
```

输入参数

• enable: bool 使能

o True: 开启软限位 o False: 关闭软限位

使用示例

```
1 arm.set_soft_limit(True)
```

各轴解锁(Unlock All Axis)

API-unlock_all_axis 各轴解锁

解除各轴锁定状态, 此API慎用。

函数原型

```
def unlock_all_axis(self)
```

使用示例

```
1 | arm.unlock_all_axis()
```

设置默认运动速度(Set Speed)

API-set_speed 设置默认运动速度

函数原型

```
1 def set_speed(self, speed):
```

输入参数

• speed: float 移动速度,单位mm/min 取值范围(0,3000],速度默认值为2000

机械臂回零(Go To Zero)

API-go_to_zero 机械臂回零

控制机械臂的各轴进入名义上的零点.

函数原型

```
1 def go_to_zero(self):
```

使用示例

```
1 arm.go_to_zero()
```

设置关节角度(Set Joint Angle)

设置关节角度

API-set_joint_angle

函数原型

```
def set_joint_angle(self, joint_angles, is_relative=False, speed=None,
wait_ok=None):
```

输入参数

• joint_angles: dict 目标关节角度字典 key是关节的ID号,取值范围[1,7]

对于关节1到关节6 value是角度(单位°),对于第七轴单位则是mm

举例: {1:45.0, 2:-30.0}

- speed: float 移动速度, 单位mm/min
- is_relative : bool 是否为相对移动
- wait_ok: bool 是否等待机械臂接收到指令后返回"ok"信息。

```
1 target_angles = {1:90.0, 2:30.0, 3:-20.0, 4:10.0, 5:0.0, 6:90.0}
2 arm.set_joint_angle(target_angles)
```

示例脚本-设置机械臂关节角度

set_joint_angle.py

```
111
   设置机械臂关节的角度,单位。
   from wlkata_mirobot import WlkataMirobot
   import time
   arm = WlkataMirobot()
7
   arm.home()
9 # 设置单个关节的角度
10 print("测试设置单个关节的角度")
11 | arm.set_joint_angle({1:100.0})
12 print ("动作执行完毕")
13 # 状态查询
   print(f"状态查询: {arm.get_status()}")
15 # 停顿2s
16 time.sleep(2)
17
18 # 设置多个关节的角度
   print("设置多个关节的角度")
19
20 target_angles = {1:90.0, 2:30.0, 3:-20.0, 4:10.0, 5:0.0, 6:90.0}
21 arm.set_joint_angle(target_angles)
22 print ("动作执行完毕")
23 # 状态查询
   print(f"状态查询: {arm.get_status()}")
25 # 停顿2s
26 time.sleep(2)
27
```

滑台控制(Slider Control)

API-set_slider_posi 设置滑台位置

函数原型

```
1 def set_slider_posi(self, d, speed=None, is_relative=False, wait_ok=True):
```

输入函数

- d: float 滑台的位置, 单位mm
- speed: float 移动速度, 单位mm/min
- is_relative: bool 是否为相对移动
- wait_ok: bool 是否等待机械臂接收到指令后返回"ok"信息。

```
print('设置滑台的位置 300mm, 速度 2000 mm/min')
arm.set_slider_posi(300, speed=2000)
```

示例脚本-滑台位置控制

slider.py

```
1.1.1
    滑台控制示例
   import time
    from wlkata_mirobot import WlkataMirobot
    print("实例化Mirobot机械臂实例")
    arm = WlkataMirobot()
8
9
10 # 机械臂本体与滑台是否同时Homing
11
   is_home_simultaneously = True
12
   # 归零
    if is_home_simultaneously:
13
        print("本体跟滑台同时归零")
14
       arm.home(has_slider=True)
15
16
   else:
17
       print("机械臂本体Homing")
18
       arm.home()
       print("滑台Homing")
19
       arm.home_slider()
20
21
    print("延时2s")
22
23
    time.sleep(2)
24
    print('设置滑台的位置 300mm, 速度 2000 mm/min')
25
26
    arm.set_slider_posi(300, speed=2000)
27
    print("延时2s")
28
29
    time.sleep(2)
30
    print('设置滑台的位置 100mm')
31
32
    arm.set_slider_posi(100)
33
    print('设置滑台的位置 相对移动 +50mm')
35
    arm.set_slider_posi(50, is_relative=True)
36
    # 更新机械臂的状态
37
38
    arm.get_status()
    print(f"当前的滑台的位置 :{arm.slider} mm")
40
```

传送带控制(Conveyor Control)

API - set_conveyor_range 设置传送带范围

函数原型

```
1 def set_conveyor_range(self, d_min=-30000, d_max=30000):
```

输入函数

- d_min: float 传送带位置最小值,单位mm 最小值不能小于-30000
- d_max: float 传送带位置最大值, 单位mm 最大值不能大于 30000

使用示例

```
1 # 设置传动带的运动范围
2 arm.set_conveyor_range(-30000, 30000)
```

API-set_conveyor-posi 设置传送带位置

函数原型

```
1 def set_conveyor_posi(self, d, speed=None, is_relative=False, wait_ok=True):
```

输入函数

- d: float 传送带的位置, 单位mm
- speed: float 移动速度, 单位mm/minis_relative: bool 是否为相对移动
- wait_ok: bool 是否等待机械臂接收到指令后返回"ok"信息。

使用示例

```
1 print('设置传送带的位置 1000mm')
2 arm.set_conveyor_posi(1000)
```

示例脚本-传送带控制

conveyor.py

```
1 ///
2 机械臂传送带示例
3 ///
4 import time
5 from wlkata_mirobot import wlkataMirobot
6
7 print("实例化Mirobot机械臂实例")
8 arm = WlkataMirobot()
9
10 # 机械臂本体与滑台是否同时Homing
11 is_home_simultaneously = True
12
13 # 注: 机械臂不需要Homing
14 print("机械臂本体Homing")
```

```
15 arm.home()
16
17
   # 设置传动带的运动范围
18 | arm.set_conveyor_range(-30000, 30000)
19
    print("延时2s")
20 time.sleep(2)
21
    print('设置传送带的位置 1000mm')
22
   arm.set_conveyor_posi(1000)
23
24
25 print("延时2s")
26
   time.sleep(2)
27
28 | print('设置传送带的位置 -3000mm')
29
    arm.set_conveyor_posi(-3000)
30
31 print("延时2s")
32
   time.sleep(2)
33
    print('设置传送带的位置 相对移动 -1000mm')
    arm.set_conveyor_posi(-1000, is_relative=True)
35
36
37
   # 更新机械臂的状态
38 arm.get_status()
    print(f"当前的传送带的的位置:{arm.conveyor} mm")
40
```

工具-选择末端工具

API-WlkataMirobotTool末端工具

在库文件的 WlkataMirobotTool 类里面定义了Mirobot机械臂的末端工具类型,继承了 Enum 类。

```
1 class WlkataMirobotTool(Enum):
2 NO_TOOL = 0 # 没有工具
3 SUCTION_CUP = 1 # 气泵吸头
4 GRIPPER = 2 # 舵机爪子
5 FLEXIBLE_CLAW = 3 # 三指柔爪
```

在使用的时候需要引入该类

```
1 | from wlkata_mirobot import WlkataMirobotTool
```

```
1 # 吸头工具
2 WlkataMirobotTool.SUCTION_CUP
```

API-set_tool_type 选择末端工具

函数原型

```
1 | def set_tool_type(self, tool, wait_ok=True):
```

输入参数

- tool: WlkataMirobotTool 末端工具对象
- wait_ok: bool 是否等待机械臂接收到指令后返回"ok"信息。

使用示例

```
1 # 更换工具-选择为吸头
2 arm.set_tool_type(WlkataMirobotTool.SUCTION_CUP)
```

示例脚本-设置末端工具类型

set_tool_type.py

```
1 | 111
   设置末端工具类型
   import time
5 | from wlkata_mirobot import WlkataMirobot, WlkataMirobotTool
7 print("实例化Mirobot机械臂实例")
8 arm = WlkataMirobot()
10 # 机械臂Home 多轴并行
11 print("机械臂Homing开始")
   arm.home()
12
   print("机械臂Homing结束")
13
14
15 # 注: 默认工具为无
16 # 状态更新与查询
17 print("更新机械臂状态")
18 arm.get_status()
   print(f"机械臂工具位姿: {arm.cartesian}")
19
20
21 # 更换工具-选择为吸头
22 | arm.set_tool_type(WlkataMirobotTool.SUCTION_CUP)
23
24 # 状态更新与查询
25 | print("更新机械臂状态")
26 arm.get_status()
27
   print(f"机械臂工具位姿(气泵): {arm.cartesian}")
28
```

工具-设置工具偏移量

API-set_tool_offset 设置工具偏移量

函数原型

```
1 def set_tool_offset(self, offset_x, offset_y, offset_z, wait_ok=True):
```

输入参数

偏移量的定义是基于腕关节坐标系原点的。

- offset_x: float X轴方向上的偏移量, 单位mm
 offset_y: float Y轴方向上的偏移量, 单位mm
 offset_z: float Z轴方向上的偏移量, 单位mm
- wait_ok: bool 是否等待机械臂接收到指令后返回"ok"信息。

使用示例

```
1 # 偏移量定义,单位mm

2 offset_x = 0

3 offset_y = 0

4 offset_z = -20.0

5 arm.set_tool_offset(offset_x, offset_y, offset_z)
```

示例脚本-设置工具的偏移量

set_tool_offset.py

```
手动设置工具坐标系的偏移量,适用于自制末端的情况。
   注: 如果是标准的末端工具,建议使用API `set_tool_type`。
4
5
   import time
   from wlkata_mirobot import WlkataMirobot, WlkataMirobotTool
6
8 print("实例化Mirobot机械臂实例")
   arm = WlkataMirobot()
9
10
11 # 机械臂Home 多轴并行
   print("机械臂Homing开始")
12
   arm.home()
13
   print("机械臂Homing结束")
14
15
16 # 注: 默认工具为无
   # 状态更新与查询
17
18 | print("更新机械臂状态")
   arm.get_status()
   print(f"机械臂工具位姿: {arm.cartesian}")
20
21
22 # 偏移量定义,单位mm
23 offset_x = 0
24 offset_y = 0
25 offset_z = -20.0
26 print(f'手动设置工具坐标系的偏移量({offset_x}, {offset_y}, {offset_z})')
   arm.set_tool_offset(offset_x, offset_y, offset_z)
27
```

```
28
29 # 状态更新与查询
30 print("更新机械臂状态")
31 arm.get_status()
32 print(f"机械臂工具位姿(气泵): {arm.cartesian}")
33
```

工具-气泵

API-pump_suction 气泵吸气

气泵吸气。如果末端接的是柔性爪,气泵吸气代表柔性爪外张。

函数原型

```
1 | def pump_suction(self):
```

使用示例

```
1 arm.pump_suction()
```

API-pump_blowing 气泵吹气

气泵吹气。如果末端接的是柔性爪,气泵吸气代表柔性爪内缩。

函数原型

```
1 def pump_blowing(self):
```

使用示例

```
1 arm.pump_blowing()
```

API-pump_off 气泵关闭

气泵关闭,。 如果末端接的是柔性爪, 气泵吸气代表柔性爪放松。

函数原型

```
1 def pump_off(self):
```

```
1 | arm.pump_off()
```

示例脚本-气泵控制

air_pump.py

```
1.1.1
1
2
    气泵控制
3
    from wlkata_mirobot import WlkataMirobot
    import time
6
7
    arm = WlkataMirobot()
    arm.home()
9
   # 气泵开启-吸气
10
11
   arm.pump_suction()
12
    # 等待5s
13
    time.sleep(5)
14
    # 气泵关闭
15
16 arm.pump_off()
17
    # 等待5s
18
    time.sleep(2)
19
   # 气泵开启-吹气
20
21 arm.pump_blowing()
22
   # 等待5s
23
   time.sleep(5)
24
   # 气泵关闭
25
26 arm.pump_off()
27
   # 等待5s
28 time.sleep(2)
```

工具-舵机夹爪

API-gripper_open 夹爪打开

夹爪打开, 此时夹爪间距为最大。

函数原型

```
1 | def gripper_open(self):
```

```
1 | arm.gripper_open()
```

API-gripper_close 夹爪闭合

夹爪闭合, 此时夹爪间距为最小。

函数原型

```
1 | def gripper_close(self):
```

使用示例

```
1 | arm.gripper_close()
```

API-set_gripper_spacing 设置夹爪间距

需要注意的是,夹爪并没有自适应抓取的功能,因此如果直接调用 gripper_close 去抓取特定的物体 会烧坏舵机。

因此更合理的是使用设置夹爪间距的API函数, 根据要抓取的目标自动调整夹爪间距。

函数原型

```
1 def set_gripper_spacing(self, spacing_mm):
```

输入参数

• spacing_mm: float 爪子间距, 单位mm

使用示例

```
1 # 设置爪子的间距
2 spacing_mm = 20.0
3 arm.set_gripper_spacing(spacing_mm)
```

示例脚本-机械臂爪相关API测试

gripper.py

```
1 | 111
2 机械臂爪相关API测试
4 from wlkata_mirobot import WlkataMirobot
   import time
5
7
   arm = WlkataMirobot()
8
   arm.home()
9
10 # 设置爪子的间距
   spacing_mm = 20.0
11
12
   arm.set_gripper_spacing(spacing_mm)
   time.sleep(2)
13
    # 爪子张开
14
```

```
arm.gripper_open()
time.sleep(2)
# 爪子闭合
arm.gripper_close()
time.sleep(2)
```

机械臂状态信息

API-get_status-获取并更新机械臂状态

更新当前机械臂的状态。在访问机械臂当前状态相关的属性之前,记得要执行该函数。

函数原型

```
1 def get_status(self, disable_debug=False):
```

使用示例

```
1 | arm.get_status()
```

机械臂状态符

获取当前机械臂状态的字符串描述。

1 | arm.status.state

• "Alarm":报警状态, 自锁状态

"Home": 归零中 "Idle": 机械臂空闲 "Busy": 机械臂运动中

关节角度

获取机械臂各个关节的角度

```
1  # 关节1的角度, 单位。
2  arm.angle.joint1
3  # 关节2的角度, 单位。
4  arm.angle.joint2
5  # 关节3的角度, 单位。
6  arm.angle.joint3
7  # 关节4的角度, 单位。
8  arm.angle.joint4
9  # 关节5的角度, 单位。
10  arm.angle.joint5
11  # 关节6的角度, 单位。
12  arm.angle.joint6
13  # 滑台的位置, 单位mm
```

工具位姿

获取工具在机械臂基坐标系下的位姿(6DoF)

```
      1
      # 工具在机械臂坐标系下的X轴坐标

      2
      arm.pose.x

      3
      # 工具在机械臂坐标系下的Y轴坐标

      4
      arm.pose.y

      5
      # 工具在机械臂坐标系下的Z轴坐标

      6
      arm.pose.z

      7
      # 工具的姿态 横滚角,单位。

      8
      arm.pose.roll

      9
      # 工具的姿态 俯仰角,单位。

      10
      arm.pose.pitch

      11
      # 工具的姿态 偏航角,单位。

      12
      arm.pose.yaw
```

气泵状态

```
1
# 电磁阀的PWM值

2
arm.valve_pwm

3
# 气泵的PWM

4
arm.pump_pwm

5
# 机械爪的PWM

6
arm.gripper_pwm
```

运动模式

```
1 # 获取机械臂的运动模式
2 # 布尔值,代表当前运动模式是相对运动还是绝对运动。
3 arm.motion_mode
```

示例脚本-获取机械臂状态

get_status.py

```
1
'''

2
获取机械臂的状态

3
'''

4
from wlkata_mirobot import wlkataMirobot

5
# 创建机械臂对象

6
arm = wlkataMirobot()

7
# 机械臂回归零点

8
arm.home()

9
# 打印机械臂当前的状态

10
print("获取机械臂的状态 ?")

11
print(arm.get_status())
```

工具-末端姿态

API-set_tool_pose

设置机械臂工具在机械臂基坐标系下的位姿。

函数原型

输入参数

```
• x:float 工具x坐标
```

• y: float 工具y坐标

• z:float 工具z坐标

• roll: float 工具横滚角

• pitch: float 工具俯仰角

• yaw: float 工具偏航角

• mode: str 运动模式

o 'p2p' 点到点快速运动

o 'linear':直线插补

• speed: float 移动速度, 单位mm/min

• is_relative : bool 是否为相对移动

• wait_ok: bool 是否等待机械臂接收到指令后返回"ok"信息。

使用示例

设置工具的坐标

```
1 | arm.set_tool_pose(200, 20, 230)
```

设置工具的坐标+欧拉角

```
1 | arm.set_tool_pose(150, -20, 230, roll=30.0, pitch=0, yaw=45.0)
```

设置工具的坐标(相对运动)

```
1 arm.set_tool_pose(5, 0, 10, is_relative=True)
```

示例脚本-机械臂工具位姿控制

set_tool_pose.py

```
1 | '''
2 机械臂工具位姿控制, 点控 point to point
3 | '''
```

```
4 from wlkata_mirobot import WlkataMirobot
   import time
   # 创建机械臂
   arm = WlkataMirobot()
   # Homing
9
   arm.home()
10
   print("运动到目标点 A")
11
   arm.set_tool_pose(200, 20, 230)
12
13
    print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
    time.sleep(2)
14
15
16
   print("运动到目标点 B")
17
   arm.set_tool_pose(200, 20, 150)
   print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
19
20
   time.sleep(2)
21
22 print("运动到目标点 C, 指定末端的姿态角")
   arm.set_tool_pose(150, -20, 230, roll=30.0, pitch=0, yaw=45.0)
   print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
24
25 time.sleep(2)
26
27 print("机械臂回零")
28 arm.go_to_zero()
```

轨迹规划-点到点(P2P)

控制机械臂以各轴设定最大速度运动;



API-p2p_interpolation

轨迹规划, 快速运动模式。 不在乎中间的轨迹。

函数原型

```
def p2p_interpolation(self, x=None, y=None, z=None, a=None, b=None, c=None, \
speed=None, is_relative=False, wait_ok=None):
```

输入参数

- x:float 工具x坐标y:float 工具y坐标
- z:float 工具z坐标
- a: float 工具横滚角

```
b: float 工具俯仰角c: float 工具偏航角
```

• speed: float 移动速度,单位mm/min

• is_relative : bool 是否为相对移动

• wait_ok: bool 是否等待机械臂接收到指令后返回"ok"信息。

使用示例

```
1 | arm.p2p_interpolation(100, 100, 150)
```

```
1 x, y, z = 100, -100, 150

2 roll, pitch, yaw = 30.0, 0, 45.0

3 arm.p2p_interpolation(x, y, z, roll, pitch, yaw)
```

示例脚本-点到点快速移动

p2p_interpolation.py

```
1 | 111
   机械臂工具位姿控制
 3 插补算法: 点到点快速移动(p2p point-to-point)
5 from wlkata_mirobot import WlkataMirobot
6
   import time
7
   # 创建机械臂
8 arm = WlkataMirobot()
9 # Homing
10 arm.home()
11
12 print("运动到目标点 A")
13 arm.p2p_interpolation(100, 100, 150)
   print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
   time.sleep(2)
15
16
17
18 print("运动到目标点 B")
19 x, y, z = 100, -100, 150
20 roll, pitch, yaw = 30.0, 0, 45.0
21
   arm.p2p_interpolation(x, y, z, roll, pitch, yaw)
22 print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
23 time.sleep(2)
```

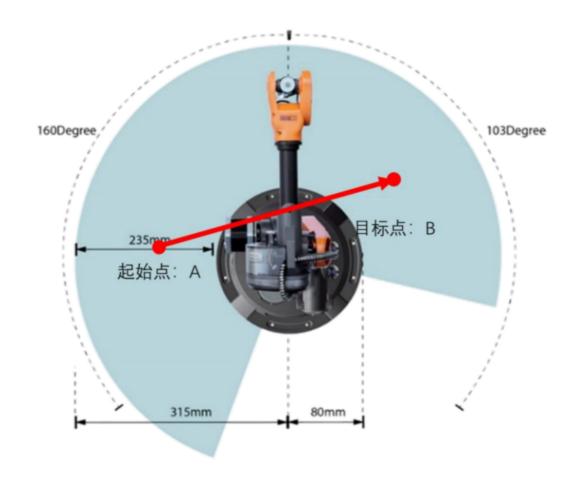
轨迹规划-直线插补

直线插补运动(末端运动轨迹为直线)



起始点: A

使用时需注意机械臂的工作空间为环形,当出现轨迹超出工作空间时,机械臂无法正常执行该指令,即使起始点 A 与目标点 B 都在工作空间内。(如下图所示),此种情况请使用P2P快速运动;



API-linear_interpolation 线性插补

函数原型

def linear_interpolation(self, x=None, y=None, z=None, a=None, b=None,
c=None, speed=None, is_relative=False, wait_ok=None):

输入参数

- x: float 工具x坐标
- y: float 工具y坐标
- z:float 工具z坐标
- a: float 工具横滚角
- b: float 工具俯仰角
- c: float 工具偏航角
- speed: float 移动速度, 单位mm/min

- is_relative: bool 是否为相对移动
- wait_ok: bool 是否等待机械臂接收到指令后返回"ok"信息。

使用示例

```
1 arm.linear_interpolation(200, -50, 150)
```

示例脚本-直线插补

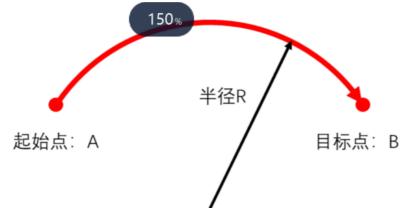
linear_interpolation.py

```
1.1.1
    机械臂工具位姿控制
   插补算法: 直线插补(linear_interpolation.)
5 from wlkata_mirobot import WlkataMirobot
   import time
6
   # 创建机械臂
7
   arm = WlkataMirobot()
   # Homing
   arm.home()
10
11
   print("运动到目标点 A")
12
    arm.linear_interpolation(200, 50, 150)
13
    print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
15
   time.sleep(2)
16
   print("运动到目标点 B")
17
    arm.linear_interpolation(200, -50, 150)
18
    print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
   time.sleep(2)
20
21
```

轨迹规划-圆弧插补

API-circular_interpolation 圆弧插补

顺时针圆弧插补/逆时针圆弧插补 (XY 平面);



在XY平面上,从当前点运动到相对坐标(ex, ey).半径为radius, 轨迹为一段圆弧, is_cw 决定圆弧是顺时针还是逆时针.

起点与终点间的距离应小于等于所设定圆弧半径值的2倍(圆弧直径)。

函数原型

```
def circular_interpolation(self, ex, ey, radius, is_cw=True, speed=None,
wait_ok=None):
```

输入参数

设置当前末端位置为圆弧起始点。

- ex:float 圆弧终止点的x坐标(相对位置),单位mmey:float 圆弧终止点的y坐标(相对位置),单位mm
- radius: float 圆弧半径,单位mmspeed: float 移动速度,单位mm/min
- wait_ok: bool 是否等待机械臂接收到指令后返回"ok"信息。

使用示例

```
1ex, ey = (0, -80)# 末端目标坐标, 单位mm(相对于当前点)2radius = 100# 半径, 单位mm3is_cw = False# 运动方向 True: 顺时针, False: 逆时针4arm.circular_interpolation(ex, ey, radius, is_cw=is_cw)
```

示例脚本-圆弧插补

circular_interpolation.py

```
1 | 111
   机械臂工具位姿控制
3 插补算法: 圆弧插补(circular interpolation.)
5 from wlkata_mirobot import WlkataMirobot
   import time
6
   # 创建机械臂
7
8 arm = WlkataMirobot()
9 # Homing
10 arm.home()
11
   print("运动到目标点 A")
12
   arm.set_tool_pose(200, 40, 150)
13
   print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
14
15
   time.sleep(2)
16
17
18
   print("运动到目标点 B(圆弧插补)")
   ex, ey = (0, -80) # 末端目标坐标, 单位mm(相对于当前点)
19
20 radius = 100
                   # 半径, 单位mm
   is_cw = False # 运动方向 True: 顺时针, False: 逆时针
21
   arm.circular_interpolation(ex, ey, radius, is_cw=is_cw)
22
23
   print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
```

```
time.sleep(2)
25
```

轨迹规划-门式轨迹规划

门型轨迹运动;



API-set_door_lift_distance 设置抬起高度

函数原型

```
1 def set_door_lift_distance(self, lift_distance):
```

输入参数

• lift_distance: float 抬起高度, 单位mm

使用示例

```
1 # 设置门式轨迹规划抬升高度
2 arm.set_door_lift_distance(50)
```

API-door_interpolation 门式轨迹插补

函数原型

输入参数

- x:float 工具x坐标
- y: float 工具y坐标
- z:float 工具z坐标
- a: float 工具横滚角
- b: float 工具俯仰角
- c: float 工具偏航角

- speed: float 移动速度, 单位mm/min
- is_relative: bool 是否为相对移动
- wait_ok: bool 是否等待机械臂接收到指令后返回"ok"信息。

使用示例

```
1 arm.door_interpolation(200, -40, 150)
```

示例脚本-门式插补

door_interpolation.py

```
1 | 111
2 机械臂工具位姿控制
3 插补算法:门式插补(door interpolation.)
4
5 from wlkata_mirobot import WlkataMirobot
6 import time
7 # 创建机械臂
8 arm = WlkataMirobot()
9 # Homing
10 arm.home()
   # 设置门式轨迹规划抬升高度
11
   arm.set_door_lift_distance(50)
13
14 print("运动到目标点 A")
   arm.set_tool_pose(200, 40, 150)
15
   print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
16
17
   time.sleep(2)
18
19
20 print("运动到目标点 B(门型插补)")
21 arm.door_interpolation(200, -40, 150)
22 print(f"当前末端在机械臂坐标系下的位姿 {arm.pose}")
23 time.sleep(2)
```